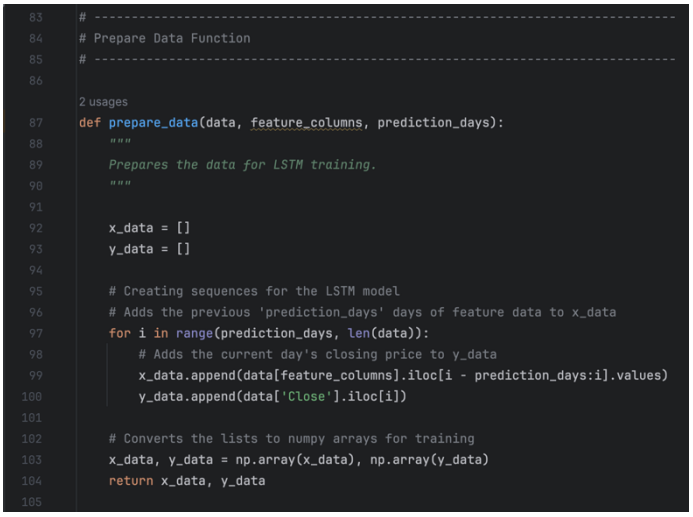
Task 3: Data processing 2

As there were a lot of difficulties trying to incorporate the requirements for the data processing in both parts 1 and 2, the decision was made to reconstruct the stock prediction file, keeping the approch closer to the original v0.1 and P1, in order to reduce confusion and lack of understanding of the code.

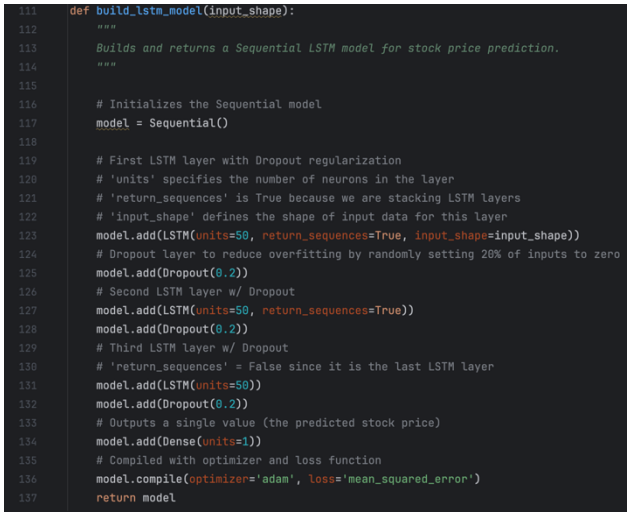
In this new version, multiple things were altered. Originally, the load data function was trying to handle too many tasks, overloaded with responsibilities and wasn't able to correctly use the feature columns, using only the 'Close' price for predictions, ignoring all the other features. This resulted with the features not being scaled properly. The proper code was added to the file, which allows for all selected feature columns to be included in the data loading and model training.

A function was also constructed for preparing the data and creating the LSTM model. This aided in code modularization, using well-defined functions to achieve their given purposes, simplifying the code. This code mostly already existed in the prior version and also v0.1 / P1, with much of it being unchanged, especially the model, since it hasn't been addressed yet in the project tasks.



```
# ------------------------------------------------------------------------------
# Prepare Data Function
# ------------------------------------------------------------------------------

2 usages
def prepare_data(data, feature_columns, prediction_days):
    """
    Prepares the data for LSTM training.
    """

    x_data = []
    y_data = []

    # Creating sequences for the LSTM model
    # Adds the previous 'prediction_days' days of feature data to x_data
    for i in range(prediction_days, len(data)):
        # Adds the current day's closing price to y_data
        x_data.append(data[feature_columns].iloc[i - prediction_days:i].values)
        y_data.append(data['Close'].iloc[i])

    # Converts the lists to numpy arrays for training
    x_data, y_data = np.array(x_data), np.array(y_data)
    return x_data, y_data

```

Figure 1 - Prepare Data Function



```
def build_lstm_model(input_shape):
    """
    Builds and returns a Sequential LSTM model for stock price prediction.
    """

    # Initializes the Sequential model
    model = Sequential()

    # First LSTM layer with Dropout regularization
    # 'units' specifies the number of neurons in the layer
    # 'return_sequences' is True because we are stacking LSTM layers
    # 'input_shape' defines the shape of input data for this layer
    model.add(LSTM(units=50, return_sequences=True, input_shape=input_shape))
    # Dropout layer to reduce overfitting by randomly setting 20% of inputs to zero
    model.add(Dropout(0.2))
    # Second LSTM layer w/ Dropout
    model.add(LSTM(units=50, return_sequences=True))
    model.add(Dropout(0.2))
    # Third LSTM layer w/ Dropout
    # 'return_sequences' = False since it is the last LSTM layer
    model.add(LSTM(units=50))
    model.add(Dropout(0.2))
    # Outputs a single value (the predicted stock price)
    model.add(Dense(units=1))
    # Compiled with optimizer and loss function
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model
```

Figure 2 - Build LSTM Model

The function's arguments included:

- data: Stock market data (DataFrame)
- feature_columns: List of column names to be used as features.
- prediction_days: Number of previous days to base predictions on.

The function returns:

- x_data, y_data: Features and labels for training/testing.

The function's arguments included:

- input_shape: The shape of the input data (for the LSTM layers).

The function returns:

- model: The compiled LSTM model.

The completion of Task 2 allowed me to progress onto Task 3, the second part of the 'Data Processing'. This required the installation of a library 'mathplotlib', allowing for some simple visualization of the results. This will allow us to "gain insights into their data and also enabling a number of AI techniques based on graphical input or image processing". The addition of Candlestick chart and Boxplot chart were outlined in the tasks.

Write a function to display stock market financial data using candlestick chart.

The implementation of a candlestick chart allows the user to identify the stock's opening and closing prices, highs and lows, and range for the given timeframe. This provides very useful data for predicting the stock price.

The function's arguments included:

- data (DataFrame): Stock market data
- n_days (int): Number of days each candlestick represents
- title (str): Title of the chart

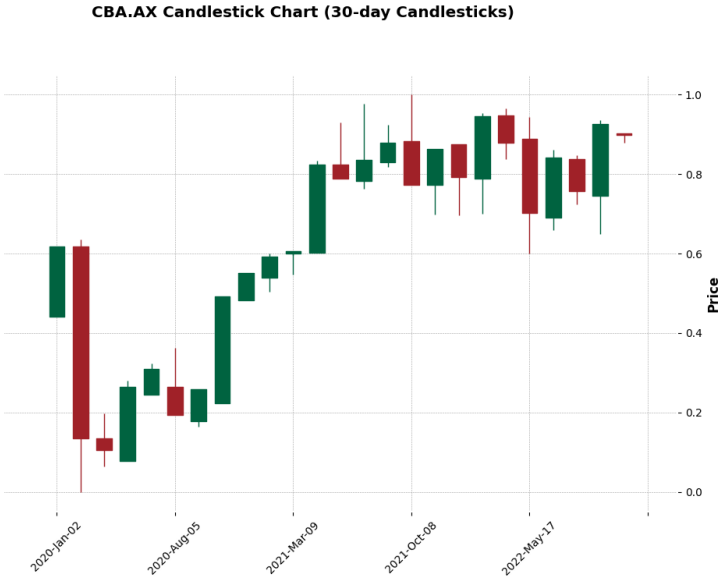
The function returns:

- None: Displays candlestick chart

The function starts out by creating a copy of the data, then uses an if statement to reset the use integer indexing for grouping the data into chunks of n_days, creating a new column 'group'. The aggregate data for each group is then collected to create the candlesticks for the data, with the date being the index of the new data frame. Then the attributes of the candlestick chart are adjusted to personal preference.

```
# ------------------------------------------------------------------------------
# Plotting Functions
# ------------------------------------------------------------------------------

def plot_candlestick(data, n_days=30, title="Candlestick Chart"):
    """
    Generates a candlestick chart over a specified number of days.
    """

    # Creates copy of data to avoid needing to modify original DataFrame
    data = data.copy()
    # Ensures data is sorted by date
    data.sort_index(inplace=True)
    data.index = pd.to_datetime(data.index)

    if n_days > 1:
        # Resets index to use integer indexing for grouping the data into chunks of n_days
        data.reset_index(inplace=True)
        # Creates a new column called 'group', assigns rows based on integer division of the index by n_days
        data['group'] = data.index // n_days

        # Aggregate data for each group to create candlesticks
        data_resampled = data.groupby('group').agg({
            'Date': 'first',
            'Open': 'first',
            'High': 'max',
            'Low': 'min',
            'Close': 'last'
        # Sets the 'Date' column as the index of the new DataFrame
        }).set_index('Date')
    else:
        # Use the original data if n_days is 1
        data_resampled = data.set_index('Date')

    # Ensures that 'Date' is datetime
    data_resampled.index = pd.to_datetime(data_resampled.index)

    # Plots the candlestick chart
    mpf.plot(
        data_resampled,
        type='candle',
        title=title,
        style='charles',
        volume=False,
        figsize=(12, 8)
    )
```

Stock Candlestick Chart Result



The candlestick chart focuses on the stock's open, high, low, and close prices. The green candles indicate periods where the stock closed higher than it opened, and the red candles show periods where it closed lower than it opened. The wicks that extend from the candles show the highest and lowest the price got to in the period of 30 days. The longer wicks at the beginning indicate higher volatility, whereas later on they appear to be shorter alongside more stable candles. The data displays a consistent upwards trend; however, declines are still present as seen by the occasional red candle.

Write a function to display stock market financial data using boxplot chart.

The implementation of the boxplot chart provides additional information for the user on the stock price, this includes key statistics about the data. It is useful for summarising the dataset and detailing the distribution of the data. If outliers exist, it can also detail them too.

The function's arguments included:

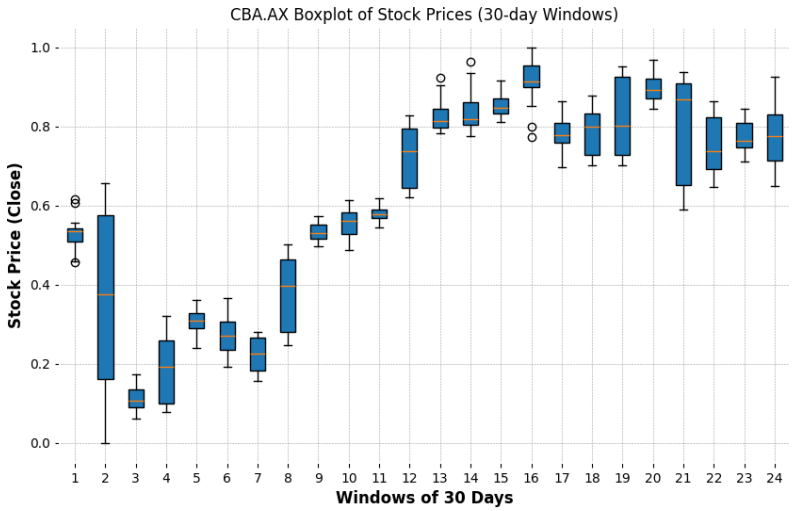
- data (DataFrame): Stock market data
- n_days (int): Number of days each candlestick represents
- title (str): Title of the chart

Returns:

- None: Displays boxplot chart

The function first segments data into 'windows' using n_days, from where it then creates a list where each element is an array of closing prices for a window. This goes for the length of the data as determined by the for loop. The 'data['Close'].iloc[i:i + n_days]' selects a small sample of the 'Close' price data. In order to only have windows suitable for plotting, a filter is created, only selecting windows with the desired size. This could potentially exclude the final window if it turns out to be too small. Then the boxplot attributes are altered to the desired outcome.

Stock Boxplot Chart Result



The boxplot displays a general upward trend, as windows later in the chart show higher stock prices, but with varying levels of volatility along the way. There also appears to be a few outliers in the first window and windows towards the middle of the curve. The windows with these outliers and large spreads indicate instability. The taller boxes earlier in the chart indicate higher volatility, with stock prices that fluctuated on a wider scale.